VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Systém pro sledování pohybu souborů a informaci

File Tracking and Information Tracking System

2012 Petr Štěpánek

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student:	**Petr Štěpánek**
Studijní program:	B2647 Informační a komunikační technologie
Studijní obor:	2612R025 Informatika a výpočetní technika
Téma:	Systém pro sledování pohybu souborů a informací File Tracking and Information Tracking System

Zásady pro vypracování:

Cílem práce je vytvořit systém umožňující monitorování přístupů k souborům na discích firemních počítačů a ukládání těchto informací na server. Tyto informace budou posléze analyzovány a využity ke zjišťování možných potencionálně nebezpečných operací, jako je například vytvoření zip souboru s citlivými firemními daty a jeho opětovné čtení při kopírování na přenosné médium. Systém bude využívat již existující ovladač, který sleduje diskové operace se soubory a zapisuje je do XML souboru. Monitorovací systém pak nahraje tyto údaje na centrální úložiště a umožní jejich další zpracování.

Systém bude obsahovat následující funkcionalitu:
1. Import XML dat na serverové úložiště.
2. Zobrazení statistických informací o využití softwaru a souborů na jednotlivých počítačích nebo pro jednotlivé uživatele.
3. Přístup k informacím bude omezen na základě oprávnění jednotlivých uživatelů. Systém bude obsahovat minimálně role (Administrátor, Manažer)
4. Implementaci pokusného algoritmu pro zjištění potencionálně nebezpečných operací.
5. Systém umožní jednoduchou rozšiřitelnost o nové analytické algoritmy.

Práce bude obsahovat:
1. Implementaci monitorovacího Informačního systému.
2. UML diagramy popisující danou implementaci.
3. Popis a dokumentaci implementovaného systému.

Seznam doporučené odborné literatury:

Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides (Gang of Four): Návrh programů pomocí vzorů. Grada. Praha 2003. ISBN 8024703025

Brett McLaughlin: Java and XML, Second Edition. O'Reilly Media, August 2001. ISBN:978-0-596-00197-1

Dále podle pokynů vedoucího bakalářské práce.

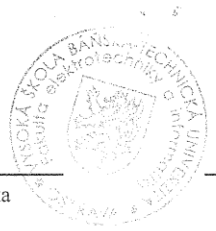
Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Martin Hanzal**

Konzultant bakalářské práce: Ing. David Ježek, Ph.D.

Datum zadání: 19.11.2010

Datum odevzdání: 06.05.2011



doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlášení studenta

Prohlašuji, že jsem tuto bakalářskou/diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

Dne: 15.5.2019 ..... Štěpánek

Podpis

4

Poděkování

Rád bych poděkoval panu Ing. Ježkovi Ph.D. za odbornou pomoc a konzultaci při vytváření této práce.

Firmě SodatSW za poskytnutí zajímavého tématu pro mou bakalářskou práci.

Prohlášení zástupce spolupracující právnické nebo fyzické osoby

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava

Dne: 2.5.2012

.....

Podpis zástupce

Abstrakt

Cílem práce je vytvořit systém umožňující monitorování přístupů k souborům na discích firemních počítačů a ukládání těchto informací na server. Tyto informace budou posléze analyzovány a využity ke zjišťování možných potencionálně nebezpečných operací jako jsou například vytvoření ZIP souboru s citlivými firemními daty a jeho opětovné čtení při kopírování na přenosné médium. Systém bude využívat prototyp ovladače, který sleduje diskové operace se soubory a zapisuje je do XML souboru. Monitorovací systém pak nahraje tyto údaje na centrální úložiště a umožní jejich další zpracování.

Klíčová slova

XML, Java, JSP, MySQL, SodatSW

Abstract

Point of this work is to be able to monitoring access to files on disk of company's computers and saving this information on server. This information will be analyzed and used to detect possibilities of dangerous operations like, for example, creating ZIP file with sensitive data and copy them on flash disk. System is using prototype of driver, which watching disk operations with files and log it into XML files. Monitoring system will upload this data on central store where they will be processed

Key words

XML, Java, JSP, MySQL, SodatSW

Seznam použitých symbolů a zkratek

XML	Extensible Markup Language (rozšiřitelný značkovací jazyk). Jazyk je určen především pro výměnu dat mezi aplikacemi a pro publikování dokumentů, u kterých popisuje strukturu z hlediska věcného obsahu jednotlivých částí, nezabývá se vzhledem.
Java	Multiplatformní objektově orientovaný programovací jazyk
JSP	Java Server Pages (Java serverové stránky). Jedná se o technologii pro tvorbu dynamických webových stránek na první pohled podobnou dobře známému PHP
MySQL	je databázový systém k dispozici jak pod bezplatnou, tak komerční placenou licencí. MySQL je multiplatformní databáze. Komunikace s ní probíhá pomocí jazyka SQL
.reg soubor	soubor který do registru Windows zapisuje nastavení a možnosti různých aplikací běžících na platformách jako kernel, SAM, drivery zařízení apod. V registrech lze také dohledat nastavení a vlastnosti systému.
Příkazová řádka	Představuje uživatelské rozhraní, ve kterém uživatel s programy nebo operačním systémem komunikuje zapisováním příkazů do příkazového řádku. Na rozdíl od textového a grafického rozhraní, nevyužívá myš ani menu a nedovede pracovat s celou plochou terminálu.
SAX	Single API for XML – typ čtení XML souboru. Soubory čte sekvenčně, jeho alternativou je DOM
DOM	Document Object Model – objektově orientovaná reprezentace XML souborů nebo HTML dokumentů. Načítá si celou strukturu a je schopen z ní číst nebo modifikovat.
GPL	GNU GPL neboli všeobecně veřejná licence GNU. Copyleftová licence která vyžaduje, aby odvozená díla byla dostupná pod toutéž licencí.
IP Adresa	Adresa internetového protokolu – číslo ve formátu A.B.C.D kde každé písmeno může nabývat hodnoty 0-255. Jednoznačně určuje počítač v počítačové síti.
JPA	Java Persistence API – Framework jazyku Java, který umožňuje snadnější práci s objekty a databázi

Checker	pracovní označení pro třídu, která kontroluje potencionální ilegální akce

Obsah

1. Úvod

Tato bakalářská práce se zabývá pasivním zabezpečením citlivých dat ve firmách. Pasivním, protože útočníkovi nezabrání ukrást data, ale pouze to ohlásí na místech, které proti útočníkovi provedou další kroky. Díky centrálnímu serveru, který jednou za nastavenou dobu přijme XML soubory ze všech stanic, které nadále zpracuje a vyhodnotí, je možné takhle hlídat i fyzicky vzdálené počítače, popřípadě notebooky s připojením na internet.

2. Driver

2.1. Původ

Driver vytvořila firma SODATSW spol. s.r.o., založena roku 1993, zabývající se ochranou dat, efektivitou práce na počítači ať už v sektoru bankovním, komerčním, stáním, sektoru samosprávy, zdravotnickém či školním. Získala různá ocenění jako třeba **The Best of Invex 2k** nebo **Křišťálový disk 2002**.

Za zmínku stojí i jejich čtyři základní certifikace:

Microsoft Partner - Silver Independent Software Vendor
ČSN EN ISO 9001:2009 – systém managementu kvality vývoje, technické podpory a implementace
ČSN ISO/IEC 27001:2006 – systém managementu bezpečnosti informací
Osvědčení podnikatele pro seznamování se s utajovanými informacemi do stupně utajení Důvěrné

Více informací na jejich webových stránkách http://www.sodatsw.cz/ .

Driver má zatím jen několik prototypových verzí, ještě není veřejně nasazen. Moje verze driveru není zdaleka nejnovější, ale pro mou práci je dostačující.

2.2. Inicializace

Inicializace driveru na jednom počítači je poměrně jednoduchá. V podstatě se skládá ze spuštění .reg souboru, následné spuštění příkazu fltmc load nasffile v příkazové řádce a spuštění testu pro kontrolu, zda se driver správně zavedl. Driver lze dodatečně nastavovat pomocí parametrů v příkazové řádce.

Data zachycené driverem získáme opět pomocí příkazové řádky a příkazu

```
nasfFileConsole.exe > NazevSouboru.xml
```

2.3. Možnosti

Mé znalosti o možnostech driveru jsou omezené pouze na takovou míru, jakou pro svou práci potřebuji. Driver jde namapovat na konkrétní složku, složky, popřípadě celý disk, nebo USB porty. V této namapované oblasti pozoruje všechny akce, jako je otevření souboru, jeho zavření, smazání, kopírování apod. a pečlivě si vše co vypozoruje, zaznamenává. Později, po nastaveném uplynulém čase, po naplnění počtu záznamů, nebo na uživatelský požadavek, driver ze svých záznamů vygeneruje xml soubor, který je následně odeslán na jiný počítač a dále zpracováván programem, který už je součástí mé bakalářské práce.

3. Klient – Server

3.1. Klient

Program klienta je nahrán na každém monitorovaném počítači. Nejvhodnější by bylo spouštět jej plánovačem úloh. Kdy už záleží na konkrétním umístění. Ve firmách např. v době kdy nikdo nepracuje, ale pouze v případě že počítače zůstávají zapnuté. Další možnost je v době obědů. Takhle by se dalo vymyslet mnoho dalších příkladů.

3.1.1. Funkce

Klient je spuštěn buď pravidelně pomocí plánovače úloh, nebo na požadavek. Nemá žádné uživatelské rozhraní, ani o sobě nedává vědět, pokud nedojde k chybě.

Pro přenos dat používá klient port 10203, a jelikož mé testování mohlo být z technických důvodu prováděno jen na jednom PC, odesílal jsem data přes localhost, takže IP klienta je nastavená na 127.0.0.1.

Pomoci funkce v mé třídy FileWorkerInformer.arrayOfFiles(String path) si získá názvy všech souborů v předem definované cestě, kterou lze definovat v konstruktoru třídy, nebo nastavením Stringové property path .

Následně otevře komunikační kanál SocketChannel , na který zareaguje server a vytvoří se spojení Client-Server. Poté se otevírají soubory, jejichž názvy nám vrátil FileWorkerInforme a odesílány vytvořeným spojením.

Soubory jsou sice odesílány postupně, ale server si je všechny, po uzavření Channelu, spojí do jednoho XML souboru.

Následně klient smaže všechny odeslané soubory a je ukončen.

3.2. Server

Server je nahrán pouze na jednom počítači, pokud možno na stejném, na kterém je databáze, aby se nemusela vytěžovat sít při nahrávání dat do databáze či získávání výsledků z dotazu na databázi

V rámci testování jsem si vytvořil na hlavní stránce tlačítko, které mi spouští server, v provozu by bylo vhodnější jiné řešení, například aby server běžel stále, nebo se spouštěl jen v době, kdy je očekáván příjem souboru.

Server je definován jako singletone, takže nehrozí jeho několika násobné spuštění, vrátí se maximálně instance, již existujícího serveru.

3.2.1. Vlastnosti serveru

Server se spustí jako vlákno, které při spuštění vytvoří a otevře ServerSocketChannel na portu 10203, a v nekonečné smyčce čeká na příchozího klienta.

V okamžiku, kdy začne komunikace Klient-Server, server spustí nové vlákno, třídy ServerConnect, předá mu příchozího klienta a na nově vytvořené vlákno si už nedrží žádnou vazbu. Sám se vrátí do stavu, kdy čeká na nového klienta.

3.2.2. ServerConnect

Třída se spouští už v samotném konstruktoru a zařizuje veškerou hlavní komunikaci s klientem.

Server třídě v konstruktoru nastaví kanál, na kterém je klient a vybere z něj naráz všechny data vložené klientem. Tímto postupem se všechny XML soubory, odeslané klientem v době poslední komunikace, spojí do jednoho XML souboru, aniž by se porušila čitelnost čí parserování dat.

Po načtení se soubor uloží do automaticky vygenerované cesty.

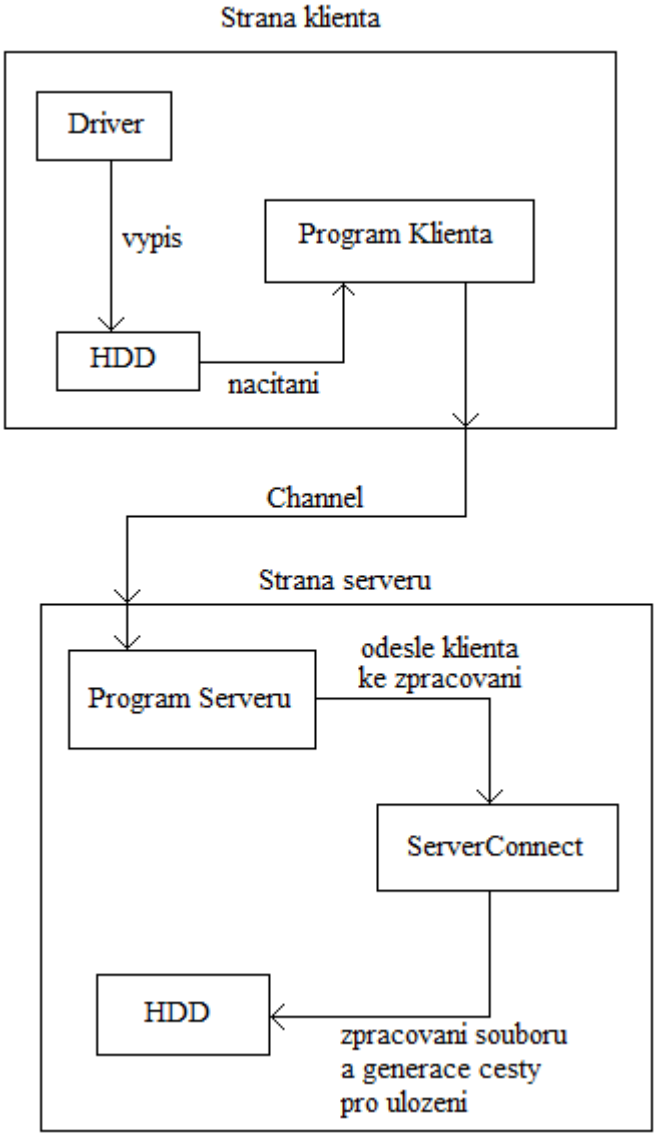
Gesta se generuje podle následujících pravidel:

Predem_nastavena_cesta\Ip-adresa\den-mesic-rok\hh_mm_ss_ms.xml

Takže výsledná cesta může být třeba následující:

C:\bakalarska prace\server\prijmuteXML\192-168-1-104\20-4-2010\14_52_31_456.xml

3.3. UML Diagram Klient-Server



Obr. 1

4. Zpracování XML souborů

4.1. Parser

Parser dědí ze třídy DefaultHandler, která je navrhnuta jako základní třída pro SAX ovladače a již v sobě má naimplementované některé základní funkce pro následné parserování.

Zpracování XML souborů je typu SAX, jelikož pro tyhle účely stačí procházet dokument postupně a okamžitě si data ukládat do správných tříd.

Pro přizpůsobení Handleru mým potřebám, jsem použil @Override na tyto metody: *startElement*, *characters* a *endElement*.

4.2. startElement

Když v XML souboru narazím na příslušný tag, označující nový záznam, tak property do které ukládám další informace o nové operaci, nastavím na null. V případě jiného tagu tato funkce nedělá nic.

4.3. characters

Ukládám si do proměnné vnitřní hodnotu mezi tagy.

Př.:
<uzivatel>Jaroslav Novák</uzivatel>

po zavolání characters

temp = Jaroslav Novák

4.4. endElement

Když v XML souboru narazím na tag, označující novou operaci, parser vytvoří novou instanci příslušné třídu, podle hexa kódu tagu.

V případě jiných tagů, plním vnitřní proměnné nové třídy daty, které jsem získal funkcí characters. Podle koncových tagu rozpoznávám, jakou vnitřní proměnnou nastavit.

Když parser dojde na tag, značící konec záznamu o operaci, uloží operaci do zásobníku operací a cyklus se opakuje, dokud se nenarazí na konec XML souboru.

Pokud by nová verze driveru měla nové tagy, program by sice oznámil chybovou hlášku, že nezná tag, ale pokračoval by dále a plnil to, co zná. Některé tagy ignoruji úmyslně, bez chybové hlášky, protože pro tuhle práci nejsou potřebné. Kdyby byli později potřeba, musí se entitním třídám přidat vhodná property ,nastavit gettery a settery na novou property a do parseru přidat řádek s příslušným tagem a setterem.

V této funkci také kontroluji, zda právě zparserovaný záznam není jen záznam o dočasném souboru, vytvořený součástí MS Office (MS Word, MS Excel…). Tyto záznamy jsou pro mě bezcenné, takže je zahodím.

endElement se také stará o převod času, který si z XML souboru načtu jako String, na formát Java.Util.Date. Tento formát používám pro snadnější získávání dat jen za určité období.

4.4.1. Náznak funkce endElement

```
@Override
public void endElement(String uri, String localName, String qName) throws SAXException {
       if(qName.equalsIgnoreCase("fat"))
       { op = createOI(temp); }  //temp byla naplnena funkci character
       else
       { filling(qName); }
   }

  private OperationsInterface createOI(String s) throws UnidentifiedOperationException
  {
     if (s.equalsIgnoreCase("2[0x2],0[0x0]"))
         {return new Cteni();}
     else if (s.equalsIgnoreCase("4[0x4],0[0x0]"))
         {return new Zapis();}
     else if (s.equalsIgnoreCase("8[0x8],0[0x0]"))
         {return new Mazani();}
                ...
     else
     { throw new UnidentifiedOperationException("CreateOI in mySAXparser.endElement: Unknown
operation//neznama operace " + s);}
   }

   private void filling(String val)
   {
            ...
      if(val.equalsIgnoreCase("PRO"))
         {op.setPRO(temp);}
      else if(val.equalsIgnoreCase("SID"))
         {op.setSID(temp);}
      else if(val.equalsIgnoreCase("DAW"))
         {op.setDAW(temp);}
      else if(val.equalsIgnoreCase("FOO"))
         {op.setFOO(temp);}
      else if(val.equalsIgnoreCase("FOP"))
         {op.setFOP(temp);}
             ...
   }
```

4.5. Zásobník operací

Během parserování souborů se pokaždé, když endElement dojde na tag značící konec operace, přidá operaci do zásobníku, který se po dokončení všech parserovacích operací, na všech souborech, odešle třídě, starající se o nahrání nových dat do databáze.

5. Databáze

5.1. Systém

Použitý databázový systém je MySQL, vytvořen firmou MySQL AB, ale nyní vlastněn společností Sun Microsystem (Dceřinou společností Oracle Corporation).

5.1.1. MySQL

Systém MySQL byl zvolen kvůli multiplatformnosti a možnosti získat jej jak pod komerční, placenou licenci, tak pod bezplatnou licencí GPL, kterou jsem pro svou práci využíval já.

Samotná komunikace probíhá v jazyce SQL.

MySQL se soustředí hlavně na rychlost a jednoduchost. Má jen jednoduché způsoby zálohování a až do nedávna nepodporovala pohledy, triggery a uložené procedury. Tyto vlastnosti se objevují až v posledních verzích. Tyhle vlastnosti, by mohli nabídnout jiné řešení mé práce, avšak já k nim neměl přístup, tudíž jsem je ani nevyužíval.

5.2. Vnitřní struktura databáze

Databáze má čtyři typy, tabulek, z toho jeden typ je celá skupina tabulek.

- První, stojící sama o sobě, bez jakékoliv vazby na ostatní tabulky, je tab. CustomUser, obsahující sloupce Name, Pass a Acces. Sloupec Acces říká, jaké oprávnění v programu má uživatel. Vice v kapitole GUI - Typy práv uživatelů.

- Druhá tabulka, FilePath, obsahuje tři sloupce, které jednoznačně určují umístění souboru na disku. A sloupec ID, který je zároveň i PK. Je využíván v poslední tabulce, označme si ho jako FP_ID. Tato tabulka je vedena jako číselník.

- Třetí tabulka SpolecneVeci obsahuje záznamy, které jsou společné pro všechny, nebo alespoň většinu operace. Například čas, ID uživatele, program, velikost souboru apod. Také obsahuje své ID, nazvané ID_Spol.

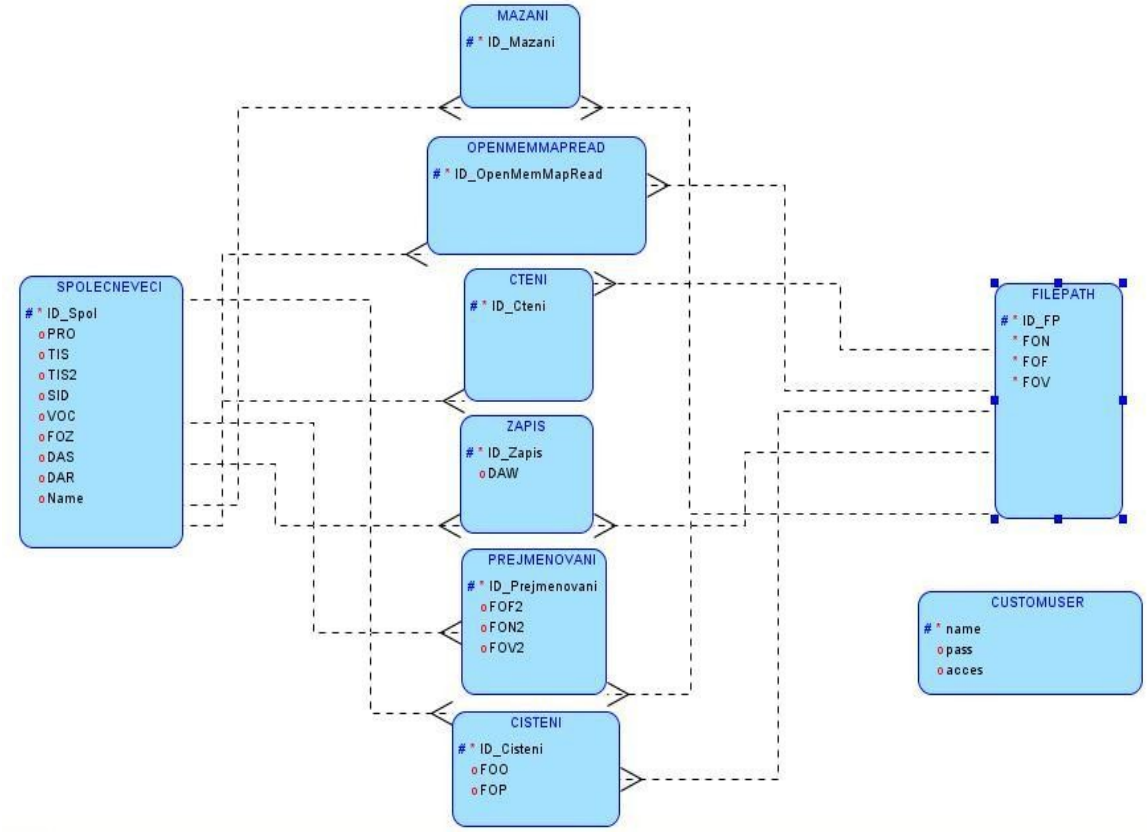
- Čtvrtá, tentokrát už skupina tabulek, jsou tabulky, jejíž názvy jsou stejné, jako operace, kterou v sobě mají uloženou. Některé z nich obsahuji dodatečné sloupce, které jsou specifické pouze pro danou operaci. Ovšem každá v sobě obsahuje dva cizí klíče, FP_ID a ID_Spol.

Touhle cestou jsem schopen jednoduše a přehledně vyhledávat v databázi potřebné informace. O veškeré spojování tabulek a generování klíčů se stara JPA. Problém nastal jedině s číselníkem, ten musím vytvářet ručně vlastní funkcí.

5.3. Datový slovník

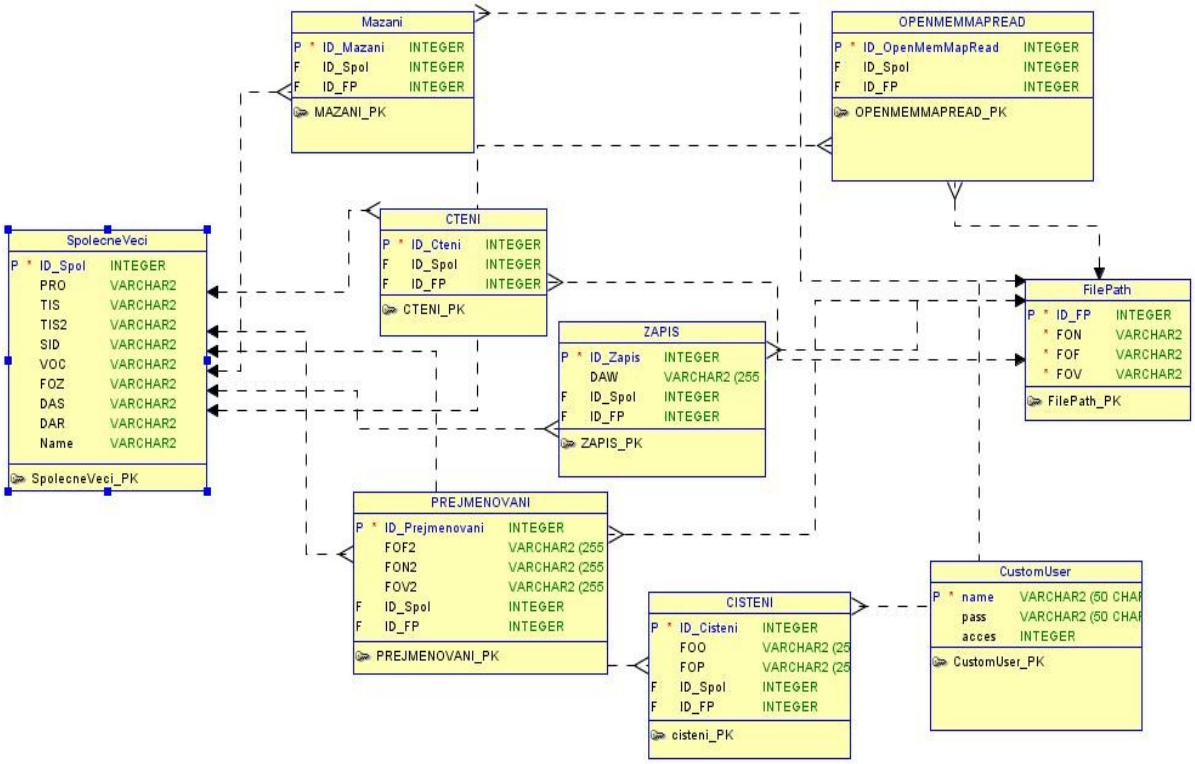
Příloha 1.

5.3.1. Logický model databáze



Obr. 2

5.3.2. Relační model databáze



Obr. 3

6. Upload do databáze

6.1. Framework

Program využívá Framework JPA a jejího providera Hibernate. Ten mapuje objekty na entity v relační databázi pomoci :

1) Mapovací soubory ve kterých je popsáno, jak se mají data transformovat do databáze a zpět.
2) Použití anotací. Např. @Id (generate = GeneratorType.AUTO) @Column (length = 30, unique = true, nullabe = false)

Každý takhle namapovaný objekt musí mít gettery, settery, hashCode() a equals ().

Díky Hibernate nemusím řešit rozdíly jednotlivých databázi. Stačí do projektu přidat jen příslušný Connector, a program bude fungovat i na jiné databázi, bez jakéhokoliv dalšího zásahu.

Objekty mapuju pomocí druhé možnosti, pro většinu proměnných mi stačí defaultně nastavené hodnoty. A i s datovým typem Date si JPA poradilo samo, pomoci jednoduché anotace

```
@Temporal(TemporalType.TIMESTAMP)
private Date TIS;     //cas zacatku
```

6.2. Connector

Connector je zapotřebí pro odesílání dotazu/aktualizací a přijímání odpovědí. V tomto případě je využit MySQL connector, který Hibernate používá pro správnou komunikaci s databází. V případě změny databáze stačí pouze stáhnout příslušný connector, nahrát ho do knihoven projektu a Hibernate bude opět schopen správně komunikovat s novou databází.

6.3. Průběh uploadu

Pro upload dat do databáze jsem z testovacích důvodů vytvořil tlačítko, které zinicializuje třídu starající se o upload a samotný upload spustí. V provozu by bylo vhodnější automatizované řešení, například ihned po přijmu dat. Pro administrátory bych tlačítko doporučil nechat, kdyby potřebovali nějaké data nahrát manuálně.

Tento proces bych rozdělil do tří akcí.

6.3.1. Vytvoření třídy a nalezení souboru

Třída XMLUploader obsahuje property path, kterou jsem programově nastavil na vhodné místo, kam server ukládá přijaté XML soubory od klientů. Stejně tak jsem programově nastavil property backupPath, vše z důvodu testování. Ale třída obsahuje i konstruktor se Stringovým parametrem path, kterým se dá nastavit, kde jsou uloženy XML soubory, stejně tak si podle zadané cesty vygeneruje cestu, kam bude ukládat zálohy XML souborů poté, co je nahraje do databáze a to v následujícím formátu.

Cesta zadaná ve Stringovém parametru

X:\bakalarska prace\prijem souboru

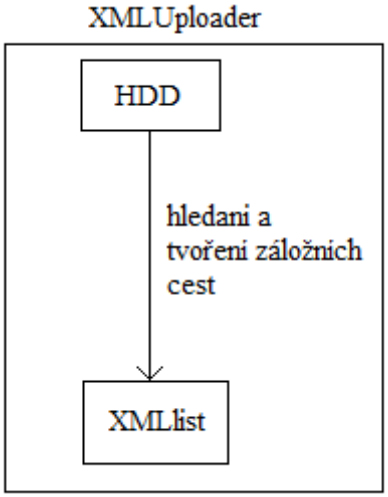
Cesta vygenerovaná pro zálohování XML souborů:

X:\bakalarska prace\zalohyXML

Třída využije stejnou třídu jakou klient, když odesílal XML soubory. Tady si akorát zjistím všechny složky (Podle pravidel generovaní to budou složky s názvy podle IP adres). Do nich se začne postupně vstupovat a opět se vytvoří seznam všech vnitřních složek (Složky s názvy podle data). V těchto složkách už jsou samotné XML soubory. Pro každý XML soubor si vytvořím instanci třídy, která určuje pozici souboru a zároveň pozici, kam má byt soubor po uploadnutí zazálohován. Tuto instanci si uložím do ArrayListu XMLlist.

Už v této době si při procházení složek s IP adresami a daty, vytvářím totožné, prázdné složky, do kterých se po uploadnutí dat, přesunou XML soubory. V případě že nahrávání dat selže, zůstanou na místě prázdné složky, díky čemuž si alespoň můžeme dohledat, kdy upload selhal a XML soubory nám zůstali na serveru (Spíše použitelné když se proměnná path a backupPath výrazně liší).

6.3.1.1. UML



Obr.4

6.3.2. Parserování a upload

Upload začíná parserováním XML souborů uložených v proměnné XMLlist. V cyklu se z cesty vytvoří soubor, který se použije jako atribut pro funkci `parsing (File target)`. Parser nemá žádný výstup, pouze si do své vnitřní property typu Stack, ukládá již hotové entity, připravené pro nahrání do databáze.

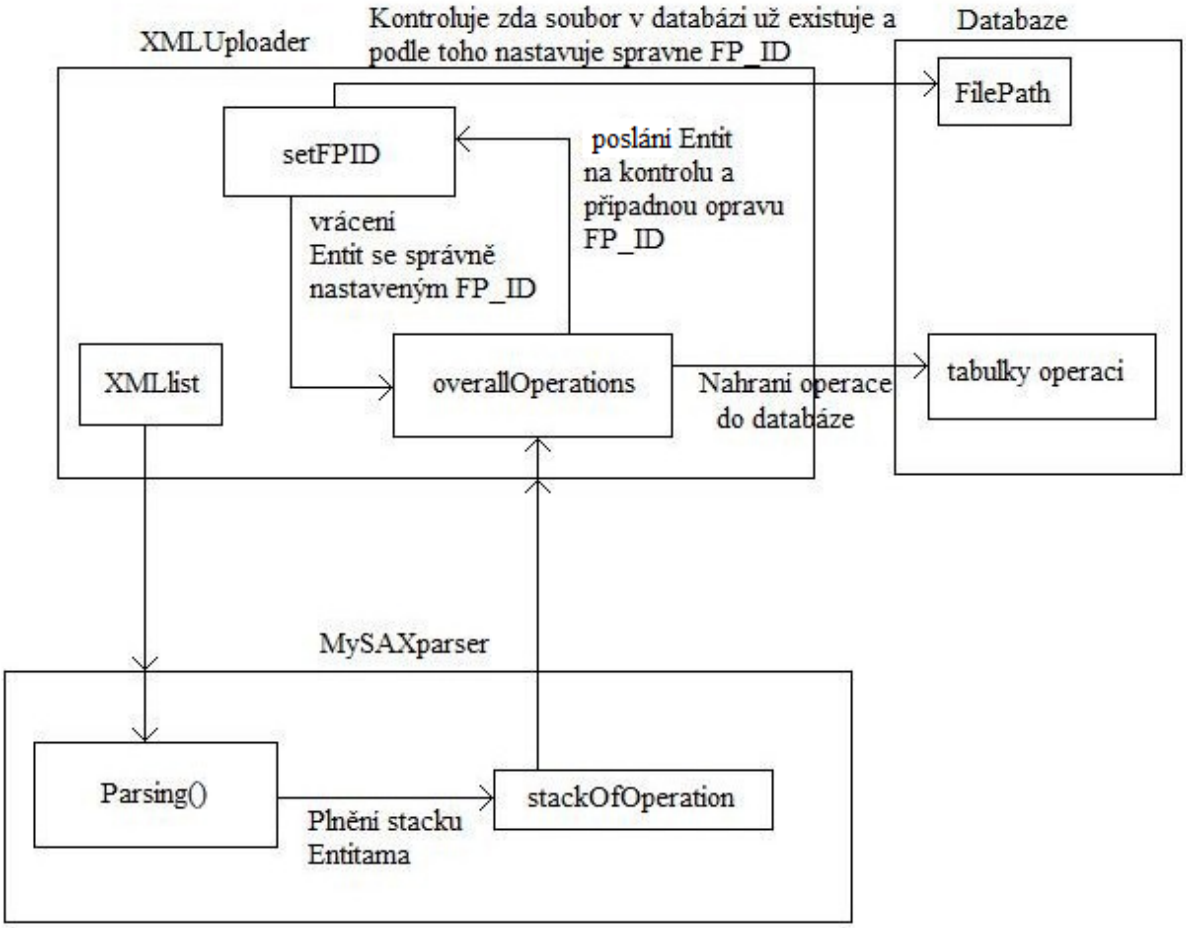
Po každém zparserovaném souboru si pomocí getu, uložím Stack do vlastní proměnné třídy XMLUploader a zásobník od parseru pro jistotu vyčistím. Takhle se cyklus opakuje, dokud je v ArrayListu ještě nějaký záznam.

Po posledním záznamu začnu projíždět Stack od XMLUploaderu, kontrolovat do jaké instance která operace patří a podle toho je posílat do databáze, ve které si to JPA už utřídí podle svých pravidel.

Pokud narazím na neznámou instanci, vzskočí mi vlastní výjimka, oznamující, jaká instance se do zásobníku dostala.

```
// OperationsInterface – interface do kterého musí spadat každá operace, abych s nimi mohl pracovat //
hromadně
for(OperationsInterface oi : overallOperations)
      {
        if (oi instanceof Cteni) {
             oi = setFPID(oi);              //setFPID je funkce, která kontroluje, zda soubor již je ---
                                            //v databázi, pokud ano, nastavím ID souboru do operace,
                                            //takhle udělám z tabulky FilePath číselník
            DAO.setCteni((Cteni)oi);
         }
        else if (oi instanceof Zapis) {
          oi = setFPID(oi);
            DAO.setZapis((Zapis)oi);
         }
        else if (oi instanceof Mazani) {
             oi = setFPID(oi);
            DAO.setMazani((Mazani)oi);
         }
        else if (oi instanceof Prejmenovani) {
             oi = setFPID(oi);
            DAO.setPrejmenovani((Prejmenovani)oi);
         }
        else if (oi instanceof OpenMemMapRead) {
             oi = setFPID(oi);
            DAO.setOpenMemMapRead((OpenMemMapRead)oi);
         }
        else if (oi instanceof Cisteni) {
             oi = setFPID(oi);
            DAO.setCisteni((Cisteni)oi);
         }
        else
        {
            System.err.println("ERROR WHILE CREATING OI CLASSES");
            throw new UnidentifiedOperationException("uploadData in FileServer.oi: Unkown
operation// neznama operace "+oi.getClass().toString()");
        }
```

6.3.2.1. UML



Obr. 5

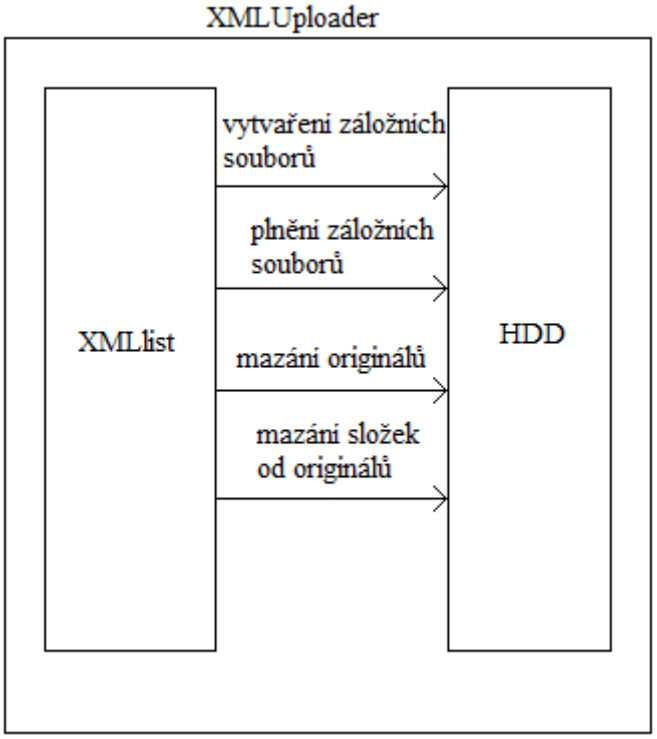
6.3.3. Zálohování a mazání

Jak bylo řečeno výše, v prvním kroku se současně vytvářeli cesty pro zálohování. Ted se opět projde XMLlist, otevřou se dva soubory. Jeden je již známé, staré XML, druhy se nachází ve složce, do které se ukládají zálohy.

Pomocí BufferedReaderu se převedou data ze zdrojového souboru do záložního a následně se zdrojový soubor smaže. Po posledním smazaném zdrojovém souboru se smažou i vygenerované složky s IP adresou a datem.

Důvodem přesunutí souboru do složky záloh je jednak možnost zpětně dohledávat informace přímo z XML souboru, či možnost opětovného nahrání dat do databáze v případě smazání databáze. Navíc odstraněním souboru ze složky přijmu zabráním opětovnému nahrání sejných dat do databáze.

6.3.3.1. UML



Obr.6

7. GUI

V následující kapitole rozeberu grafické rozhraní, funkce různých prvků a programové řešení akcí vyvolaných GUI prvky. Grafické rozhraní se zobrazuje v internetovém prohlížeči. Osobně jsem celí program testoval v prohlížeči od organizace Mozilla, a jejím produktu FireFox v. 11.0.

Prvky a funkce v grafickém rozhraní pochází ze tří Java Server knihoven, které na začátek stránky importujeme pomocí příkazu taglib.

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">
<%@taglib prefix="f" uri="http://java.sun.com/jsf/core"%>
<%@taglib prefix="h" uri="http://java.sun.com/jsf/html"%>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
```

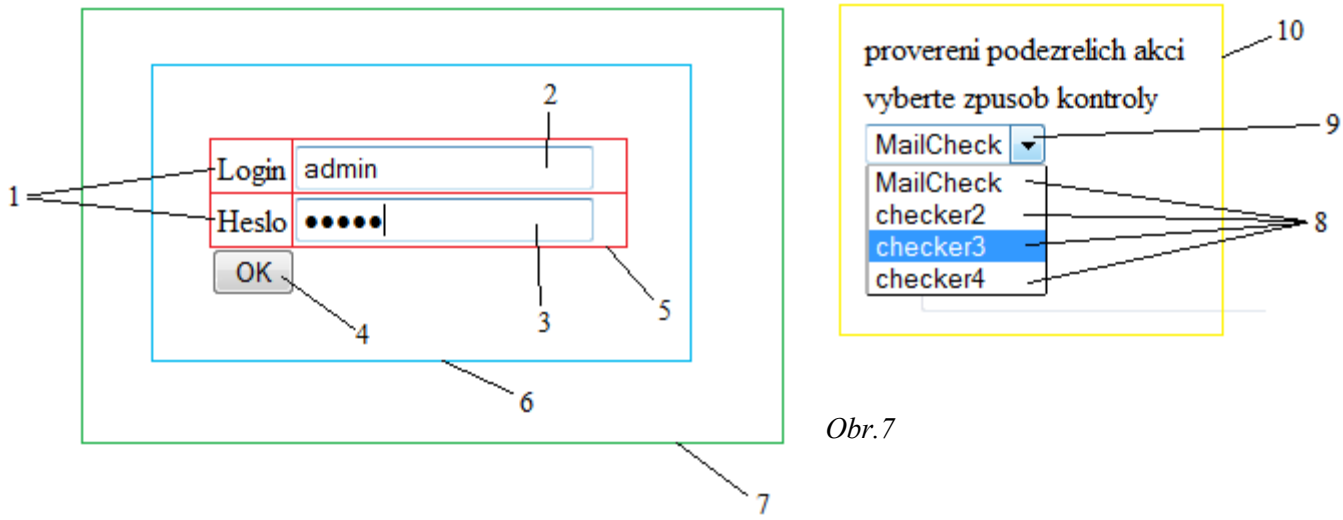
7.1. Tabulka využití prefixů

Prefix f	
Tag	Popis tagu
<f:view></f:view>	Slouží jako kontejner pro ostatní JavaServer tagy
<f:selectItems/>	Použit v kombinaci s <h:selectOneMenu>. Naplní výběrové
	menu, položkami z listu, určeného hodnotou value

Prefix h	
Tag	Popis tagu
<h:form>	komponenta, která slouží jako kontejner pro ostatní komponenty,
	schopné shromaždovat, nebo zobrazovat data od uživatele
<h:outputLabel/>	Komponenta, jejíž Stringová hodnota value se zobrazí uživateli
<h:panelGrid></h:panelGrid>	párový tag, který urovnává vnitřní komponenty do pomyslné mřížky
<h:commandButton/>	Příkazové tlačítko, obvykle po zmáčknutí volá nějakou funkci v programu
<h:inputText/>	Komponenta, díky které se uživateli zobrazí pole pro napsání vlastního textu,
	je provázána s vnitřní proměnou ManagedBean třídy, do které se text ukládá
<h:inputSecret/>	Stejné jako inputText, jen místo napsaných znaků zobrazuje ****, vhodné pro
	zadávání hesla
<h:graphicImage/>	Pro zobrazení obrázku v prohlížeči
<h:selectOneMenu>	Zobrazí rozevírací menu, známe taky pod názvem comboBox. Položky z menu
	se plní tagem <f:selectItems/> popsaným výše.
	Vybraná položka se uloží do proměnné ManagedBeany

Prefix c	
Tag	Popis tagu
<c:if></c:if>	Slouží stejně jako if v Javě. V tomto programu kontroluje oprávnění uživatele.
	V podstatě čím vyšší má uživatel oprávnění, tím více může vidět komponent

7.1.1. Příklady prefixů



Obr.7

1) h:outputLabel
2) h:inputText
3) h:inputSecret
4) h:commandButton
5) h:panelGrid
6) h:form
7) f:view
8) f:selectItems
9) h:selectOneMenu
10) c:if

7.2. Filter

Můj filtr implementuje rozhraní Filter, které se používá ve webových aplikacích například k následujícím účelům:

1) Autentizace uživatele
2) Logovaní a kontrola účtů
3) Konverze obrázků
4) Komprese dat
5) Kódování dat
6) Kontrola tokenem

…

Tady se filtr používá pro autorizaci uživatelů a umožnění jim přistup na stránky. Ze začátku jsou všechny stránky pod filtrem, který zabraňuje jakémukoliv pokusu o zobrazení. Jediná stránka vyjmuta z filtru je Login stránka, kde se uživatel může přihlásit do systému, čímž získá a nastaví instanci uživatele, díky které pak může vstoupit kamkoliv do systému. Co na stránkách uvidí je ovšem omezeno jeho oprávněním a tagem <c:if>. V případě odhlášení uživatele se zruší uživatelská instance a správně se zobrazí pouze Login stránka.

V případě pokusu o zobrazení jiné než logovací stránky, bez přihlášeného uživatele, se zobrazí následující stránka s chybou 403.

Obr.8

7.2.1. Funkce <c:if>

Jak bylo řečeno výše, tento tag, uložený v online knihovně http://java.sun.com/jsp/jstl/core, funguje stejně jako if v Javě, a v systému jej využívám jako podpůrný prostředek k filtru, na rozlišení oprávnění uživatelů. Kdo má jaké oprávnění a čím je rozlišuji, naleznete v následující kapitole Typy práv uživatelů.

Když se uživatel přihlásí svým loginem a heslem z databáze, vrátí se z databáze i číslo, určující na jaké úrovni oprávnění uživatel je a podle toho vidí dané komponenty v systému, které mu umožňují provádět různé operace. Podívejme se na příklad na *Obr.7*, tuto část systému může vidět pouze vedení a samozřejmě administrátor systému, jelikož vedení má číslo oprávnění 2 a administrace 3, bude kód stránky vypadat následovně.

```
<c:if test="${sessionScope['logged'].acces >= 2}">  // sessionScope['logged'] je instance přihlášeného
                                                    // uživatele
        <h:panelGrid columns="1">
                <h:outputText value="provereni podezrelich akci"/>
                <h:outputText value="vyberte zpusob kontroly"/>
                <h:selectOneMenu id="selectOneWay" value="#{dbComm.way}">
                        <f:selectItems value="#{dbComm.wayList}" />
                </h:selectOneMenu>
                <h:commandButton value="Check" action="#{dbComm.illegalActionCheck}"/>
        </h:panelGrid>
</c:if>
```

7.2.2. Typy práv uživatelů a jejich možnosti v systému

Práva se dělí do tří základních skupin označeny čísly. Předpokládané hiearchistické rozdělení ve firmě, tudíž i v tomto systému je zaměstnanec, management, vedení a administrace systému. Při vytváření systému jsem nepředpokládal, že by řadový zaměstnanec měl mít do tohoto systému přístup, aby třeba kontroloval sám sebe, byl v řešení zcela vypuštěn a nejnižší stupeň oprávnění má management firmy.

Každá skupina vidí i to, co vidí skupiny s nižší hodnotou oprávnění

7.2.2.1. Management podniku

Management podniku má hodnotu oprávnění 1. Vše co si mohou v systému prohlédnout je graf, zobrazující využití softwaru ve firmách, podle čehož se mohou případně rozhodnout, jestli dokupovat licence některých programů, zda lidi raději nepoužívají nějaké freeware programy, zároveň v něm jde poznat, jak moc se na pracovních počítačích hrají hry. Menší problém se může zdát ten, že názvy programů se zobrazují jako .exe soubory (Například spuštění hry Counter-Strike se bude zobrazovat jako hl.exe). Ve skutečnosti to není až takový problém, protože firma SodatSW má rozsáhlý soubor, nebo možná databázi .exe souborů a k ním příslušný název programu. Takže stačí pouze tyto dvě věci propojit.

Graf lze zaměřit na celou firmu, ale také jej můžeme omezit na určitého uživatele, určité datum nebo rozsah mezi daty, popřípadě kombinaci omezení uživatele a data.

Také zde je možnost odhlásit se ze systému.

7.2.2.2. Vedení podniku

Tato skupina má hodnotu oprávnění 2 a kromě práce s grafem, už může kontrolovat nebezpečné operace se soubory. Jelikož možností znehodnocení či odcizení citlivých firemních dat je určitě mnoho, musí existovat i mnoho způsobu kontroly těchto ilegálních akci. Proto si člen vedení vybere v rozklikávacím menu způsob kontroly souboru a spustí test. V této verzi je naimplementovaný pouze jeden způsob kontroly.

Po dokončení testu se vedení zobrazí tabulka potencionálně nebezpečných souboru a vedle každého záznamu tlačítko, které zjistí historii souboru na disku. Tyhle informace by pro vedení měli být dostačující pro to, aby věděli, zda proti zaměstnanci nějak zakročit či ne.

7.2.2.3. Administrace systému

Administrátorům byla udělena hodnota oprávnění 3 a mají nad systémem úplnou kontrolu. Opět mají samozřejmě přístup k funkcím nižších skupin, ale taky získali následující možnosti.

Přidání nového uživatele do systému. Vkládají jméno, heslo a stupeň oprávnění

Editace config souboru přímo ze systému. V dosavadní verzi jsou tři config soubory. První v sobě skrývá názvy způsobu kontroly souborů. Když se do systému přidá nový způsob kontroly, tzv. „checker“, a prováže se s položkou z combo boxu, administrátoři (samozřejmě to jde udělat už hned při updatu programu), mohou změnit název položky v combo boxu, popřípadě přidat další položky. Toto je také výhodné když některé Checkery nechcete používat, tímto je jednoduše vyřadíte ze systému.

Editace souboru s internetovými prohlížeči. Momentálně jediný existující checker používá tento soubor, ve kterém jsou uloženy .exe soubory internetových prohlížečů (firefox.exe, explorer.exe, opera.exe…). Checker tyto názvy potřebuje pro svou práci, proto když vznikne nový prohlížeč, nebo starý změní název svého .exe souboru, musí se tento soubor příslušně editovat.

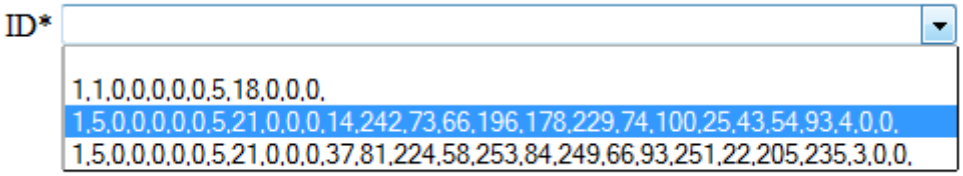
Editace souboru s komprimátory. Slouží stejně jako předchozí soubor, akorát k němu není vytvořený žádný checker, který by jej využíval.

7.3. Popis GUI a funkčnost jednotlivých prvků

V předchozí kapitole byla nastíněna funkčnost jednotlivých částí programu, dále rozeberu programové řešení prvku

7.3.1. Manažerská část

Obr. 9



Obr. 10

Na Obr. 9 vidíme část programu, určenou pro management podniku. První komponenta je combo box, ve kterém si uživatel vybere SID zkoumaného uživatele. Pokud chceme zkoumat všechny uživatele, zvolíme první, prázdnou hodnotu. Pro tyto případy by bylo vhodné v databázi vytvořit tabulku jmen uživatelů a jejich SID. V combo boxu pak zobrazovat pouze jména uživatelů. SID se nahrávají do boxu v momentě, kdy se uživatel přihlásí do systému.

Po zvolení SID zaměstnance se nastaví zkoumané datum, pole pro vkládání jsou omezená na počet znaků dva, nebo čtyři, podle toho zda zadáváme den, měsíc nebo rok. Program není ošetřen, pokud zadáme nesmyslné data jako 51.18. 2040. I přes to program najde záznamy v zadaném rozmezí.

Po stisku tlačítka “Vyuziti softwaru” program vezme nastavené hodnoty a vytvoří z nich datumy typu GregorianCalendar. Zkontroluje, zda byl nastaveny SID a podle toho zavolá funkci s omezením na uživatele nebo bez omezení. Obě funkce jsou stejné, jen funkce bez omezení nemá v dotazu na databázi omezení where, proto budu popisovat pouze funkci s omezením na SID.

V zavolané funkci vytvořím v entityManageru následující dotaz

```
"select PRO, count(PRO), TIS from SpolecneVeci where sid = '" + id + "' group by PRO"
```

A do listu vložím list výsledku. Takto vytvořený list pošlu do vlastní funkce, která mi vrácený list typu Object[], překonvertuje na List typu Program. Tento list ještě vložím do funkce, která mi něj odstraní všechny objekty, nevyhovující zadanému datovému rozmezí. Takto hotový list se teprve vrací z entityManageru zpět do ManagedBeany.

Následně se vytvoří servlet, kterému se nastaví atributy obsahující právě získaný list a SID uživatele (pokud SID nebylo zadáno, nastaví se Stringová hodnota „Vsechno“) a přesmerujem se na JSP stránku, která slouží, pomocí tagu <h:graphicImage/> k zobrazení grafu, hodnota tagu odkazuje na servlet.

Tlačítko Log Out pouze zruší instanci přihlášeného uživatele, a odstraní atribut logged v sessione. Což má za následek opětovnou aktivaci filtru, který vám nezobrazí žádnou stránku, dokud se znova nepřihlásíte.

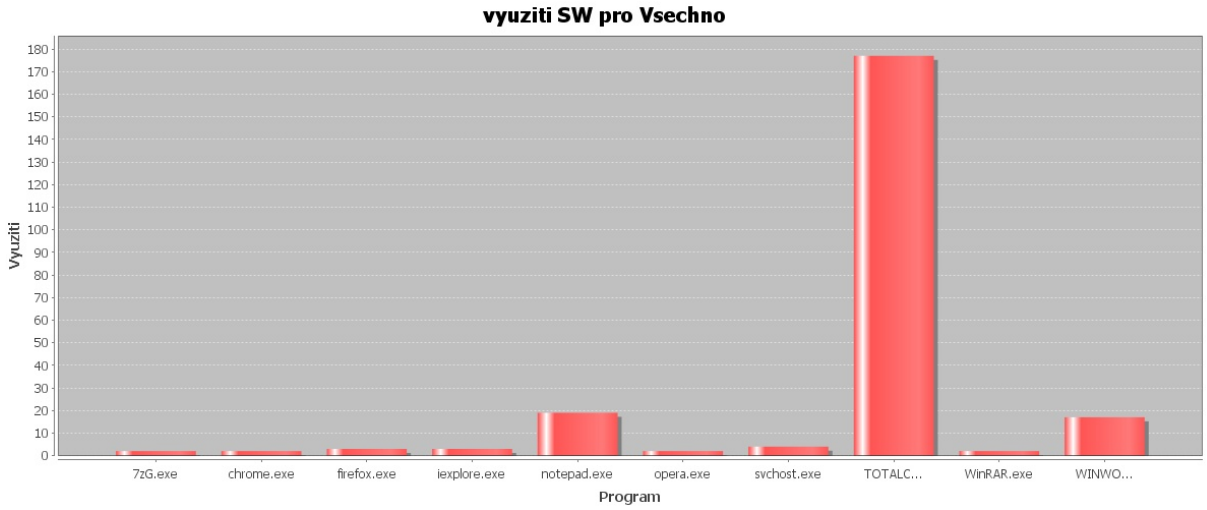
7.3.1.1. Servlet

Má dvě vnitřní, statické, finální proměnné WIDTH a HEIGHT, pro nastavení výšky a délky grafu.

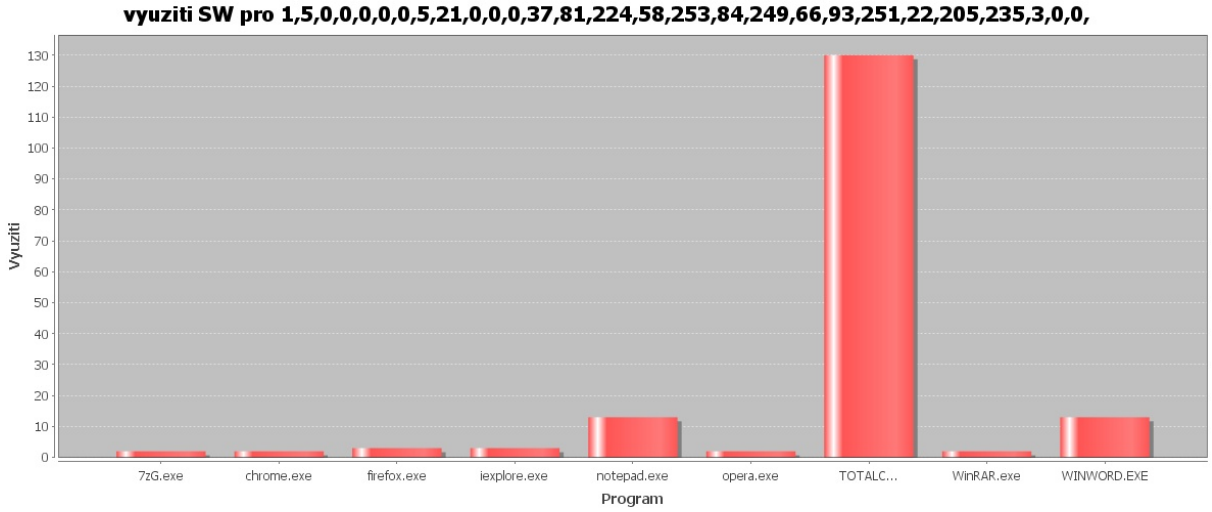
@Override jsem použil pouze na metodu doGet, která se volá když potřebuju ze servletu něco získat. V tomto případě získat a zobrazit jim vytvořený graf. Ostatní metody jsem nechal bez povšimnutí.

V metodě doGet nejdříve nastavím návratový typ na image/jpg. Poté vložím dříve nastavené atributy do nově vytvořených proměnných. Všechny servletové proměnné vložím do funkce paintGraph mé třídy graphPainting.

Funkce paintGraph využívá dodatečně importované knihovny org.jfree.chart, která je specializovaná pro tvorbu různých typů grafů (koláčové, sloupcové…) a její distribuce je zdarma. Jako graf pro zobrazení využití softwaru jsem zvolil standartní, sloupcoví graf.

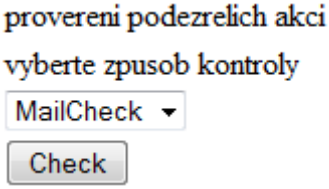


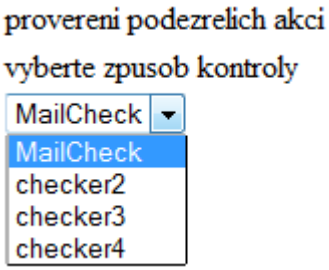
Obr. 11



Obr. 12

7.3.2. Část pro vedení





Obr 13. *Obr 14.*

Vedení může obsluhovat i část programu, která umožňuje kontrolovat zaměstnance a citlivé soubory firmy. V combo boxu si vedoucí vybere způsob kontroly souborů. Kontrola není omezená na SID zaměstnanců, protože v případě že se ilegální akce účastní více než jeden zaměstnanec, stal by se Checker neefektivní. Po určení způsobu kontroly, jej potvrdíme tlačítkem „Check“, které spustí kontrolu v následujícím sledu.

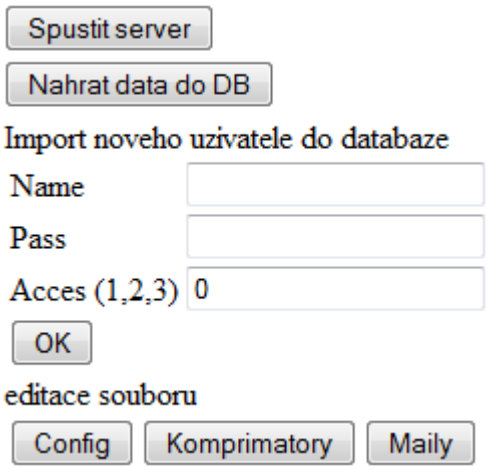
- Zkontroluje Stringový název vybrané kontroly a porovná jej se všemi názvy v combo boxu, podle toho určí do kterého ifu má pokračovat.

```
String [] array = new String [wayList.size()];
 wayList.toArray(array);
        if (way.equalsIgnoreCase(array[0]))
        { //Checker na první pozici }
        else if(way.equalsIgnoreCase(array[1]))
        { //Checker na druhé pozici }
        else if(way.equalsIgnoreCase(array[2]))
        {       …
```

- Vytvoří se instance Checkeru, spadající pod interface checkerI, jehož jediná povinnost je mít metodu public List<OperationsInterface> check(SessionDBMySQLocal sl), která se taky následně zavolá.
- Člen vedení je následně přesměrován na JSP stránku s výsledky. Na které je vedle každého výsledku tlačítko, pro zjištění historie souboru na disku, která se zobrazí na další JSP stránce.

V kapitole 8. se podrobněji podíváme, co se stane, když se jako způsob kontroly vybere MailCheck.

7.3.3. Administrátorská část



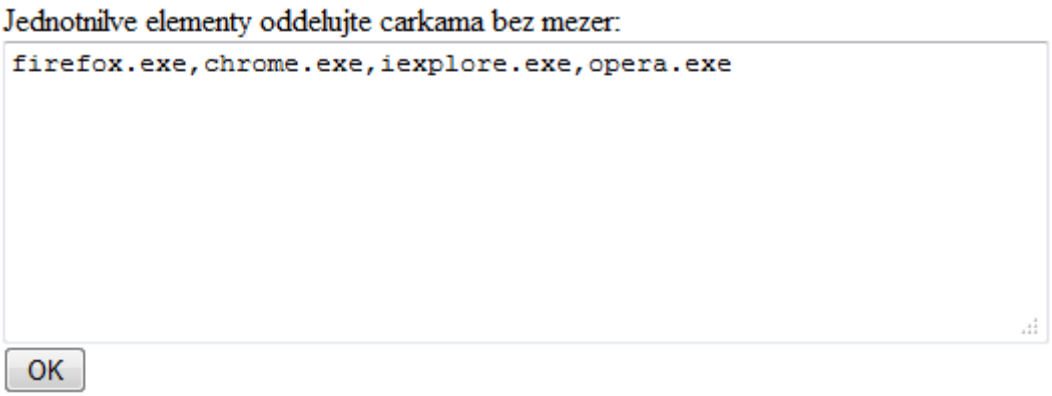
Obr. 15

Tato část je z velké části popsána napříč celé dokumentace. Tlačítko „Spustit server“ je pro manuální spuštění serveru, jehož funkce jsou popsány v kapitole 3.2

Tlačítko „Nahrat data do DB“ opět slouží k manuálnímu nahrání dat, přijaté serverem, do databáze. Tady bych se rád odkázal na kapitolu 6.3

Import nového uživatele do databáze probíhá jednoduchým vložením nové entity do databáze, kde položka „Name“ musí být originální, jelikož je vedena jako primární klíc. O rozdělení práv uživatelů (Položka „Acces“) pojednává kapitola 7.2.2

Poslední tři tlačítka se opět odkazují na poslední tři odstavce kapitoly 7.2.2.3. GUI pro každé tlačítko je zobrazeno na *Obr. 16*, pouze text v oblasti s názvy Checkeru/prohlížečů/komprimátorů se liší, podle příslušného tlačítka.



Obr. 16

Stiskem tlačítka „OK“ se uloží příslušný soubor. Tyto tři soubory jsou uloženy v balíčku s Checkerama a jejich interfacem.

8. Checkery

Z anglického slova check – dozor, kontrola, prověrka. Slouží k samotné kontrole zaměstnanců a souborů na PC a noteboocích. Jeho funkce je výhradně pasivní a pouze upozorňuje na potencionálně nebezpečné akce na firemních discích a externích zařízeních. Zákrok proti zaměstnanci musí nakonec provést pověřená osoba, za předpokladu, že uzná upozornění za pravděpodobné, jelikož každý checker může vyhodnotit neškodnou akci za nebezpečnou.

Třídy s novými Checkery musí spadat pod interface checkerI jehož jedinou podmínkou je funkce

```
public List<OperationsInterface> check(SessionDBMySQLocal sl);
// parameter sl v sobě obsahuje EentityManagera, který je pro chod checkeru nezbytný
```

8.1. MailCheck

Původně konstruován na hledání souboru odesílané pomoci e-mailu. Zaměřen na e-mailové schránky, do kterých přistupujeme přes internetové prohlížeče. Při prvních testech jsem zjistil, že kontroluje, i když soubory nahrávám na internetová úložiště (např. www.ulozto.cz, www.leteckaposta.cz apod.). Není zaměřen na poštovní klienty typu Thunderbird a nebyl na nich testován, i když pomocí možnosti editace Config souboru by se na něj Checker mohl jednoduše rozšířit.

Rozšíření MailChecku o možnost výpisu historie života souboru na disku zlepšuje jeho možnosti, jak lze vidět v kapitole 8.1.2

8.1.1. Základní funkce

Po vybrání tohoto Checkeru se vytvoří jeho nová instance a z configu se mu nastaví jména internetových prohlížečů. Poté se zavolá zmiňovaná funkce check(SessionDBMySQLocal sl).

Postupně se v cyklu prochází list internetových prohlížečů a jejich název se dává do podmínky where v dotazu na databázi. Databáze se ptá výhradně tabulky entit typu Cteni, jelikož jiné akce internetových prohlížečů pro mě, v tomto případě nejsou podstatné. Odpovědi z databáze se postupně ukládají do listu. Když se vrátí odpověď na poslední typ internetového prohlížeče a přidá se do listu, celý list se vrátí do ManagedBeany, která uživatele přesměruje na stránku s výsledky. Viz *Obr. 17*

Nazev souboru	Nazev programu	SID uzivatele	Datum a cas provedeni akce	
TAJNE ODSUD.DOCX	firefox.exe	1,5,0,0,0,0,0,5,21,0,0,0,37,81,224,58,253,84,249,66,93,251,22,205,235,3,0,0,	2012-02-25 16:44:54.0	Historie souboru
TAJNE Z NETU.DOC	firefox.exe	1,5,0,0,0,0,0,5,21,0,0,0,37,81,224,58,253,84,249,66,93,251,22,205,235,3,0,0,	2012-02-25 16:45:01.0	Historie souboru
DOVOLENA.JPG	firefox.exe	1,5,0,0,0,0,0,5,21,0,0,0,37,81,224,58,253,84,249,66,93,251,22,205,235,3,0,0,	2012-04-24 17:32:25.0	Historie souboru
TAJNE Z NETU.DOC	chrome.exe	1,5,0,0,0,0,0,5,21,0,0,0,37,81,224,58,253,84,249,66,93,251,22,205,235,3,0,0,	2012-02-25 16:46:26.0	Historie souboru
TAJNE ODSUD.DOCX	chrome.exe	1,5,0,0,0,0,0,5,21,0,0,0,37,81,224,58,253,84,249,66,93,251,22,205,235,3,0,0,	2012-02-25 16:46:37.0	Historie souboru
TAJNE ODSUD.DOCX	iexplore.exe	1,5,0,0,0,0,0,5,21,0,0,0,37,81,224,58,253,84,249,66,93,251,22,205,235,3,0,0,	2012-02-25 16:48:58.0	Historie souboru
TAJNE Z NETU.DOC	iexplore.exe	1,5,0,0,0,0,0,5,21,0,0,0,37,81,224,58,253,84,249,66,93,251,22,205,235,3,0,0,	2012-02-25 16:49:06.0	Historie souboru
TAJNE Z NETU.DOC:ZONE.IDENTIFIER	iexplore.exe	1,5,0,0,0,0,0,5,21,0,0,0,37,81,224,58,253,84,249,66,93,251,22,205,235,3,0,0,	2012-02-25 16:49:06.0	Historie souboru
TAJNE ODSUD.DOCX	opera.exe	1,5,0,0,0,0,0,5,21,0,0,0,37,81,224,58,253,84,249,66,93,251,22,205,235,3,0,0,	2012-02-25 16:53:25.0	Historie souboru
TAJNE Z NETU.DOC	opera.exe	1,5,0,0,0,0,0,5,21,0,0,0,37,81,224,58,253,84,249,66,93,251,22,205,235,3,0,0,	2012-02-25 16:53:31.0	Historie souboru

Obr. 17

V této tabulce jasně vidíme kdy, kdo, jak a co odeslal přes internet pryč. Když se zadíváme na řádek tři, vidíme, že byl pomoci programu FireFox odeslán obrázek s názvem „Dovolena.jpg“. Kdybychom měli jakékoliv pochybnosti, použijeme tlačítko „Historie Souboru“, abychom se ujistili, zda někdo opravdu jen odesílal fotku z dovolené.

8.1.2. Rozšíření o historii souboru

Při hledání historie souborů nás zajímají všechny akce se souborem, dokud nezměnil jméno. Poté musíme v databázi najít nové ID přejmenovaného souboru a rekurzivně tuto činnost opakovat, dokud existuje stále nějaké přejmenování.

Ze začátku si vrátím informace o souboru na vybraném řádku a zavolám přetíženou metodu trackFile, jako parametr ji dám informace o souboru. Metoda poté zavolá sama sebe, akorát jako druhý parametr dá nové, aktuální datum.

Do databáze pošlu dotaz na jméno souboru, který se po přejmenování jmenoval Dovolena.jpg, ve složce X, na disku Y (Jméno složky a disku je drženo v listu, zobrazeném v tabulce, i když je nevidíme). Nyní nastávají dvě možnosti.

1) V databázi není nalezena žádná položka. To znamená, že soubor nebyl přejmenováván, a já zachytím výjimku NoResultException, v ní nastavím příznak, značící, že se v databázi nic nenašlo a program pokračuje ve finally do správné if části. V té pošlu dotaz na všechny tabulky, aby mi vrátili záznamy, s ID souborem dovolené.jpg. Poté se výsledný list protáhne funkci, která z něj smaže všechny záznamy nevyhovující časovému rozmezí. V tomto případě by se nesmazalo nic, jelikož defaultně je časové rozmezí nastaveno na nejnižší hodnotu a aktuální datum. Tato funkce je zde kvůli rekurzi, která nastane v druhém případě. Tento list je pro přehlednost, pomocí algoritmu quickSort setřízen, podle času operací a vrácen do ManagedBeany a zobrazí se na obrazovce. V tomto případě může výsledný list vypadat podobně, jako na *Obr. 18*.

Nazev souboru	Program	SID	Cas	Operace
_DEVICE_HARDDISKVOLUME2 _NASFFILETEST_DCKO_PRACOVNI VECI_ TAJNE ODSUD.DOCX	firefox.exe	1,5,0,0,0,0,0,5,21,0,0,0,37,81,224,58,253,84,249,66,93,251,22,205,235,3,0,0,	2012-02-25 16:44:54.0	Cteni
_DEVICE_HARDDISKVOLUME2 _NASFFILETEST_DCKO_PRACOVNI VECI_ TAJNE ODSUD.DOCX	chrome.exe	1,5,0,0,0,0,0,5,21,0,0,0,37,81,224,58,253,84,249,66,93,251,22,205,235,3,0,0,	2012-02-25 16:46:37.0	Cteni
_DEVICE_HARDDISKVOLUME2 _NASFFILETEST_DCKO_PRACOVNI VECI_ TAJNE ODSUD.DOCX	iexplore.exe	1,5,0,0,0,0,0,5,21,0,0,0,37,81,224,58,253,84,249,66,93,251,22,205,235,3,0,0,	2012-02-25 16:48:58.0	Cteni
_DEVICE_HARDDISKVOLUME2 _NASFFILETEST_DCKO_PRACOVNI VECI_ TAJNE ODSUD.DOCX	opera.exe	1,5,0,0,0,0,0,5,21,0,0,0,37,81,224,58,253,84,249,66,93,251,22,205,235,3,0,0,	2012-02-25 16:53:25.0	Cteni
_DEVICE_HARDDISKVOLUME2 _NASFFILETEST_DCKO_PRACOVNI VECI_ TAJNE ODSUD.DOCX	TOTALCMD.EXE	1,5,0,0,0,0,0,5,21,0,0,0,37,81,224,58,253,84,249,66,93,251,22,205,235,3,0,0,	2012-03-31 14:25:29.0	Cteni
_DEVICE_HARDDISKVOLUME2 _NASFFILETEST_DCKO_PRACOVNI VECI_ TAJNE ODSUD.DOCX	WinRAR.exe	1,5,0,0,0,0,0,5,21,0,0,0,37,81,224,58,253,84,249,66,93,251,22,205,235,3,0,0,	2012-03-31 14:25:49.0	Cteni
_DEVICE_HARDDISKVOLUME2 _NASFFILETEST_DCKO_PRACOVNI VECI_ TAJNE ODSUD.DOCX	7zG.exe	1,5,0,0,0,0,0,5,21,0,0,0,37,81,224,58,253,84,249,66,93,251,22,205,235,3,0,0,	2012-03-31 14:26:45.0	Cteni
_DEVICE_HARDDISKVOLUME2 _NASFFILETEST_DCKO_PRACOVNI VECI_ TAJNE ODSUD.DOCX	WINWORD.EXE	1,5,0,0,0,0,0,5,21,0,0,0,37,81,224,58,253,84,249,66,93,251,22,205,235,3,0,0,	2012-03-31 14:32:12.0	OpenMemMapRead
_DEVICE_HARDDISKVOLUME2 _NASFFILETEST_DCKO_PRACOVNI VECI_ TAJNE ODSUD.DOCX	TOTALCMD.EXE	1,5,0,0,0,0,0,5,21,0,0,0,37,81,224,58,253,84,249,66,93,251,22,205,235,3,0,0,	2012-03-31 14:33:29.0	Cteni

Zpet na hlavi stranku

Obr. 18

2) V databázi je nalezeno jméno souboru, ze kterého bylo přejmenováno na Dovolena.jpg. V takovém případě program pokračuje do druhé if části. V té pošlu dotaz na všechny tabulky, aby mi vrátili záznamy, s ID souborem dovolené.jpg. Poté se výsledný list protáhne funkci, která z něj smaže všechny záznamy nevyhovující časovému rozmezí, kdy při prvním projití je spodní hodnota čas, kdy byl soubor přejmenován a horní hodnota aktuální datum a čas. Takto odstraním z listu falešné záznamy, které mohou vzniknout například smazáním jednoho souboru a přejmenováním druhého souboru na stejné místo se stejným jménem. Výsledné data se uloží do listu, které se bude vracet co ManagedBeany, a zavolá se metoda rekurzivně. První parametr se nastaví soubor, který byl přejmenován na Dovolená.jpg a druhý parametr bude spodní omezující datum, které v rekurzivní metodě bude sloužit jako horní omezující datum. Takto metoda volá opakovaně sama sebe, dokud nenastane případ, kdy nastane možnost 1) Výsledný list pak bude vypadat takto.

Nazev souboru	Program	SID	Cas	Operace
_DEVICE_HARDDISKVOLUME2 _NASFFILETEST_DCKO_PRACOVNI VECI_111_ FIREMNI DATA.DOCX	TOTALCMD.EXE	1,5,0,0,0,0,0,5,21,0,0,0,37,81,224,58,253,84,249,66,93,251,22,205,235,3,0,0,	2012-04-24 17:28:42.0	Zapis
_DEVICE_HARDDISKVOLUME2 _NASFFILETEST_DCKO_PRACOVNI VECI_111_ FIREMNI DATA.DOCX	TOTALCMD.EXE	1,5,0,0,0,0,0,5,21,0,0,0,37,81,224,58,253,84,249,66,93,251,22,205,235,3,0,0,	2012-04-24 17:28:42.0	Cisteni
_DEVICE_HARDDISKVOLUME2 _NASFFILETEST_DCKO_PRACOVNI VECI_111_ FIREMNI DATA.DOCX	TOTALCMD.EXE	1,5,0,0,0,0,0,5,21,0,0,0,37,81,224,58,253,84,249,66,93,251,22,205,235,3,0,0,	2012-04-24 17:29:25.0	Prejmenovani
_DEVICE_HARDDISKVOLUME2 _NASFFILETEST_DCKO_PRACOVNI VECI_111_ ULOZISTE1.DOCX	TOTALCMD.EXE	1,5,0,0,0,0,0,5,21,0,0,0,37,81,224,58,253,84,249,66,93,251,22,205,235,3,0,0,	2012-04-24 17:29:29.0	Prejmenovani
_DEVICE_HARDDISKVOLUME2 _NASFFILETEST_DCKO_PRACOVNI VECI_222_ ULOZISTE1.DOCX	TOTALCMD.EXE	1,5,0,0,0,0,0,5,21,0,0,0,37,81,224,58,253,84,249,66,93,251,22,205,235,3,0,0,	2012-04-24 17:29:36.0	Prejmenovani
_DEVICE_HARDDISKVOLUME2 _NASFFILETEST_DCKO_PRACOVNI VECI_222_ ULOZISTE2.DOCX	WINWORD.EXE	1,5,0,0,0,0,0,5,21,0,0,0,37,81,224,58,253,84,249,66,93,251,22,205,235,3,0,0,	2012-04-24 17:30:33.0	OpenMemMapRead
_DEVICE_HARDDISKVOLUME2 _NASFFILETEST_DCKO_PRACOVNI VECI_222_ ULOZISTE2.DOCX	WINWORD.EXE	1,5,0,0,0,0,0,5,21,0,0,0,37,81,224,58,253,84,249,66,93,251,22,205,235,3,0,0,	2012-04-24 17:30:43.0	OpenMemMapRead
_DEVICE_HARDDISKVOLUME2 _NASFFILETEST_DCKO_PRACOVNI VECI_222_ ULOZISTE2.DOCX	TOTALCMD.EXE	1,5,0,0,0,0,0,5,21,0,0,0,37,81,224,58,253,84,249,66,93,251,22,205,235,3,0,0,	2012-04-24 17:30:52.0	Prejmenovani
_DEVICE_HARDDISKVOLUME2 _NASFFILETEST_DCKO_PRACOVNI VECI_333_ ULOZISTE2.DOCX	TOTALCMD.EXE	1,5,0,0,0,0,0,5,21,0,0,0,37,81,224,58,253,84,249,66,93,251,22,205,235,3,0,0,	2012-04-24 17:30:57.0	Prejmenovani
_DEVICE_HARDDISKVOLUME2 _NASFFILETEST_DCKO_PRACOVNI VECI_333_ ULOZISTE3.DOCX	WINWORD.EXE	1,5,0,0,0,0,0,5,21,0,0,0,37,81,224,58,253,84,249,66,93,251,22,205,235,3,0,0,	2012-04-24 17:31:00.0	OpenMemMapRead
_DEVICE_HARDDISKVOLUME2 _NASFFILETEST_DCKO_PRACOVNI VECI_333_ ULOZISTE3.DOCX	TOTALCMD.EXE	1,5,0,0,0,0,0,5,21,0,0,0,37,81,224,58,253,84,249,66,93,251,22,205,235,3,0,0,	2012-04-24 17:31:13.0	Prejmenovani
_DEVICE_HARDDISKVOLUME2 _NASFFILETEST_DCKO_PRACOVNI VECI_444_ ULOZISTE3.DOCX	TOTALCMD.EXE	1,5,0,0,0,0,0,5,21,0,0,0,37,81,224,58,253,84,249,66,93,251,22,205,235,3,0,0,	2012-04-24 17:31:21.0	Prejmenovani
_DEVICE_HARDDISKVOLUME2 _NASFFILETEST_DCKO_PRACOVNI VECI_555_ ULOZISTE3.DOCX	TOTALCMD.EXE	1,5,0,0,0,0,0,5,21,0,0,0,37,81,224,58,253,84,249,66,93,251,22,205,235,3,0,0,	2012-04-24 17:31:37.0	Prejmenovani
_DEVICE_HARDDISKVOLUME2 _NASFFILETEST_DCKO_PRACOVNI VECI_555_ DOVOLENA.JPG	firefox.exe	1,5,0,0,0,0,0,5,21,0,0,0,37,81,224,58,253,84,249,66,93,251,22,205,235,3,0,0,	2012-04-24 17:32:25.0	Cteni

Zpet na hlavi stranku

Obr. 19

Tabulka je celkem jasná a přehledná, a lze na první pohled vidět, že soubor Dovolená.jpg, ve skutečnosti je .docx dokument s firemními daty, které byli tímto způsobem odeslány ven. Jen bych rád objasnil první sloupec tabulky, ten se může zdát nepřehledný, jelikož jsem z důvodu parserování a pozdějších dotazů na databázi ze strany Javy, musel zaměnit znak '\' za znak '_'. Proto bych rád ukázal skutečnou cestu, alespoň prvního záznamu.

_DEVICE_HARDDISKVOLUME2
_NASFFILETEST_DCKO_PRACOVNI VECI_111_
FIREMNI DATA.DOCX

První řádek je název disku. Driver neukládá jméno disku pod písmeny, ale pod systémovým názvem. Druhý řádek jsou názvy složek a poslední je samotný název souboru. Nesmíme zapomenout na nahrazení '_' za znak '\'. Takže pro nás lépe čitelná, výsledná cesta je tato

C:\NASFFILETEST\DCKO\PRACOVNI VECI\111\FIREMNI DATA.DOCX

9. Závěr

9.1. Zhodnocení dosažených výsledků

Porovnám li dosažené výsledky se zdáním

1) Import XML dat na serverové úložiště – pomocí vlastní síťové architektury Klient – Server, posílám data na serverové úložiště, aniž by program Klienta o sobě dával graficky vědět. Jen výstupný XML soubor nesmí ve jméně obsahovat mezeru (Z důvodu dalšího parserování a práce se souborem).
2) Zobrazení statických informací o využití softwaru a souborů na jednotlivých počítačích, nebo pro jednotlivé uživatele – Manažerská část IS pomocí grafu, generovaného servletem, zobrazuje informace o využití softwaru pro jednotlivé či všechny uživatele. Pro zobrazování těchto informaci pro jednotlivé počítače mi driver nedává potřebná data a za předpokladu že jsou IP adresy přidělovány dynamicky např. přenosným počítačům, by byl graf nepřesný, kdyby se měl orientovat podle IP adres programu klienta, který mi odesílá XML soubory. Graf pro využití souborů mi přišel zbytečný a jeho implementace by byla obdobná jako implementace grafu o využití softwaru.
3) Přístup k informacím bude omezen na základě oprávnění jednotlivých uživatelů, Systém bude obsahovat minimálně role Administrátor, Manažer – Omezení přístupu jsem docílil pomocí Filtru a možnosti c:if na JSP stránkách. Systém obsahuje tři role, Administrátor, Vedení a Manažer.
4) Implementace pokusného algoritmu pro zjištění potencionálně nebezpečných operací – Naimplementoval jsem algoritmus MailCheck, který dokáže zjistit potencionálně nebezpečné operace a dokáže najít historii vybraného souboru.
5) Systém umožní jednoduchou rozšiřitelnost o nové analytické algoritmy – V případě přidání nového kontrolního algoritmu stačí nastavit config soubor a dle potřeby v kódu inicializovat třídu nového algoritmu.

9.1.1. Vlastní přínos

Pro firmu jsem otestoval prototyp jejího driveru, a upozornil jsem na několik věcí, např.

- Jakým způsobem probíhá například akce mazání souborů(V podstatě dojde pouze k přejmenování kdy výsledná cesta je koš)
- Některým operacím chybí například SID uživatele, což pro mě nemělo vliv, ale jiné Checkery by toho mohli využívat.
- Driver je schopný držet informace i o souborech se streamem ZONE.IDENTIFIER (Stream označující že soubor nepochází z tohoto počítače). Což může otevřít možnosti pro další využití.

Navíc jim předám třídu MailCheck s příslušnými dotazy do databáze. Kterou mohou zdokonalit či na jejím základě vytvořit další kontrolní algoritmy.

9.2. Zhodnoceni z pohledu dalšího vývoje

V případě nasazení do reálného provozu, by musel být systém schopen automaticky vybírat z driveru XML soubory a ty automaticky odesílat na server, který by byl buďto pořád spuštěný, nebo by spouštěl v určitou dobu.

V případě velkého množství Checkeru zautomatizovat jejich spouštění a pověřenou osobu případně upozorňovat, třeba ikonou v system trayi. Taky by bylo vhodné vytvořit něco jako ignore list pro akce klasifikované jako ilegální, které ve skutečnosti nepodstatné, aby je příští kontrola už znova zbytečně nezobrazovala.

Pro zvýšení přehlednosti by bylo vhodné napojit systém na firemní databázi zaměstnanců se jmény a SID čísli, abychom nemuseli ještě dohledávat jméno zaměstnance, který pravděpodobně poškodil firmu.

Dalším kontrolní algoritmy my se mohli naučit pracovat s pravděpodobností. Např. když se komprimují soubory a do toho si zaměstnanec otevře pro čtení další soubor. Jaká je šance,že byl soubor zkomprimován do balíku, či s balíkem nemá nic společného a byl pouze otevřen.

Možnost sledování dat v reálném čase, už při parserování XML souboru si držet aktuální pozici a vlastnosti souboru, být stále schopen dohledat historii souboru na disku a poradit si i s problémem držení informaci o souborech v reálném čase v případě že se soubor zduplikuje a každá kopie povede jinou cestou.

Zjistit co se stane v případě, že ze dvou různých PC bude do databáze odeslán soubor se stejným jménem a cestou. Tuto možnost jsem z technických důvodů nemohl otestovat.

9.2.1. Návaznost na projekty řešené na pracovišti

Již to bylo zmíněno v práci, ale tento systém by se určitě dal propojit s databází programů a jejich .exe souborů, pro zvýšení přehlednosti grafů a sledovacích tabulek. Jelikož se firma zabývá i personálním auditem a efektivitou práce ve firmách, mohla by se manažerská část programu rozšířit o tyto jejich specializované programy, které určitě poskytnou více informací než můj jednoduchý graf.

Firma se zabývá také šifrováním dat. Teoreticky, když se vytvoří správný Checker, mohl by kontrolovat, zda zaměstnanci používají šifrování dat, jak mají.

Bohužel do firmy, jejich projektech a stylu, jakým jejich programy fungují, nevidím dostatečně a je možné že tyhle rozšíření jsou pro ně zbytečná, nebo je už vyřešené mají.

9.3. Zdroje

http://www.sodatsw.cz
http://tutorials.jenkov.com/
http://download.oracle.com/javase/1.4.2/docs/api/
http://www.stackoverflow.com
http://www.hibernate.org
http://exadel.com/web/portal/jsftags-guide
http://www.algoritmy.net/

10. Seznam příloh

10.1. Příloha 1 - Datový slovník

FilePath

Název	Typ	Klíč	Null	popis
ID_FP	Int	PK	ne	
FON	VARCHAR2		ano	Soubor
FOF	VARCHAR2		ano	Složka
FOV	VARCHAR2		ano	Disk

SpolecneVeci

Název	Typ	Klíč	Null	popis
ID_Spol	VARCHAR2	PK	ne	
PRO	VARCHAR2		ano	Program
TIS	VARCHAR2		ano	Čas začátku operace
TIS2	VARCHAR2		ano	Čas konce operace
SID	VARCHAR2		ano	ID uživatele
VOC	VARCHAR2		ano	Zařízení (HDD, flash…)
FOZ	VARCHAR2		ano	Velikost souboru
DAS	VARCHAR2		ano	Počet čtených dat
DAR	VARCHAR2		ano	Kolikrát se čte, kolik se přečetlo

Mazani

Název	Typ	Klíč	Null	popis
ID_Mazani	Int	PK	ne	
ID_Spol	Int	FK	ne	Cizí klíč k datům operace
ID_FP	Int	FK	ne	Cizí klíč k souboru operace

OpenMemMapRead

Název	Typ	Klíč	Null	popis
ID_OpenMemMapRead	Int	PK	ne	
ID_Spol	Int	FK	ne	Cizí klíč k datům operace
ID_FP	Int	FK	ne	Cizí klíč k souboru operace

Cteni

Název	Typ	Klíč	Null	popis
ID_Cteni	Int	PK	ne	
ID_Spol	Int	FK	ne	Cizí klíč k datům operace
ID_FP	Int	FK	ne	Cizí klíč k souboru operace

Zapis

Název	Typ	Klíč	Null	popis
ID_Zapis	Int	PK	ne	
ID_Spol	Int	FK	ne	Cizí klíč k datům operace
ID_FP	Int	FK	ne	Cizí klíč k souboru operace
DAW	VARCHAR2		ano	Kolikrát se zapisuje, kolikrát se zapsalo

Prejmenovani

Název	Typ	Klíč	Null	popis
ID_OpenMemMapRead	Int	PK	ne	
ID_Spol	Int	FK	ne	Cizí klíč k datům operace
ID_FP	Int	FK	ne	Cizí klíč k souboru operace
FON2	VARCHAR2		ano	Cílový soubor
FOF2	VARCHAR2		ano	Cílová složka
FOV2	VARCHAR2		ano	Cílový disk

Cisteni

Název	Typ	Klíč	Null	popis
ID_Mazani	Int	PK	ne	
ID_Spol	Int	FK	ne	Cizí klíč k datům operace
ID_FP	Int	FK	ne	Cizí klíč k souboru operace
FOO	VARCHAR2		ano	Originální velikost souboru
FOP	VARCHAR2		ano	Ukazatel na FileObject